Supporting Shared Resource Usage for a Diverse User Community: the OSG Experience and Lessons Learned

Gabriele Garzoglio[1]*, Tanya Levshina*, Mats Rynge‡, Chander Sehgal*, Marko Slyz*

* Computing Sector, Fermi National Accelerator Laboratory, Batavia, IL
E-mail:{garzoglio, tlevshin, cssehgal, mslyz}@fnal.gov

‡ Information Sciences Institute (ISI), Marina del Rey, CA
E-mail: rynge@isi.edu

Abstract. The Open Science Grid (OSG) supports a diverse community of new and existing users in adopting and making effective use of the Distributed High Throughput Computing (DHTC) model. The LHC user community has deep local support within the experiments. For other smaller communities and individual users the OSG provides consulting and technical services through the User Support area. We describe these sometimes successful and sometimes not so successful experiences and analyze lessons learned that are helping us improve our services. The services offered include forums to enable shared learning and mutual support, tutorials and documentation for new technology, and troubleshooting of problematic or systemic failure modes. For new communities and users, we bootstrap their use of the distributed high throughput computing technologies and resources available on the OSG by following a phased approach. We first adapt the application and run a small production campaign on a subset of "friendly" sites. Only then do we move the user to run full production campaigns across the many remote sites on the OSG, adding to the community resources up to hundreds of thousands of CPU hours per day. This scaling up generates new challenges – like no determinism in the time to job completion, and diverse errors due to the heterogeneity of the configurations and environments – so some attention is needed to get good results. We cover recent experiences with image simulation for the Large Synoptic Survey Telescope (LSST), small-file large volume data movement for the Dark Energy Survey (DES), civil engineering simulation with the Network for Earthquake Engineering Simulation (NEES), and accelerator modeling with the Electron Ion Collider group at BNL. We will categorize and analyze the use cases and describe how our processes are evolving based on lessons learned.

1. Introduction

The Open Science Grid (OSG) is a consortium of more than 100 institutions including universities, national laboratories, and computing centers. OSG fosters scientific research and knowledge supporting the computational activities of more than 80 communities. [1] Because in the OSG model resources are federated, when some resources are idle at a particular institution they can be used by other communities. This use of resources is unscheduled and, therefore, referred to as *opportunistic*. The OSG User Support group helps communities port their computing operations to OSG opportunistic resources.

[1] To whom any correspondence should be addressed.

OSG primarily supports running many simultaneous jobs that don't need low-latency communication to other jobs. This paradigm is called Distributed High Throughput Computing (DHTC). Some applications can't easily be made to fit this paradigm, and may run better on supercomputers. Other applications are more naturally suitable for DHTC. Porting these applications to the OSG consists in the determining the best sequence of Grid service invocations and processing steps, called a *workflow*, that allow the application to best exploit the OSG resources.

The capabilities of the standard Grid infrastructure will handle many applications using a simple, commonly used workflow. [2] At the other extreme, some applications may have complicated requirements that should be handled with specialized software. [3] This paper describes the porting of applications to use Grid workflows ranging from low to medium complexity. These might serve as patterns for porting applications with similar requirements.

2. Outline of Resources and Software Available on OSG

The OSG computing infrastructure consists of a collection of institutions, called *sites*. Each typically has one or more cluster of computers. There is significant latitude in how OSG sites are configured, but nevertheless some environments are more common than others.

2.1. Job Management

Sites expose their resources for external access through standard Grid interfaces, in particular GRAM. [4] There are several tools that allow the use of these resources; however, because of its flexibility and reliability, Condor-G [5] is among the most used for submitting and managing jobs on the OSG.

Condor is often used in conjunction with the glidein Workload Management System (glideinWMS) [6]. Using Condor-G to interact with the sites, glideinWMS submits placeholder jobs, or *pilots*, that reserve worker nodes for a specific community. This mechanism effectively creates a virtual batch system that provides transparent access to computers at diverse sites throughout the Grid. This is called an *overlay* batch system since it runs on the various batch systems present at the sites.

There are a large number of applications that Condor and glideinWMS support with no special effort. Among the services available, Condor provides mechanisms to transfer data. Because of the typical load on OSG servers and associated networks, these mechanisms seem to work best if the amount of data is less than about 1 GB per job. To overcome this limitation we adopt other methods, discussed in the next sections.

2.2. Data Transfer Methods

One of the main non-Condor-based data transfer methods is gridFTP. [7] This service adds Grid security on top of the ftp transfer protocol. More than ftp and other standard protocols such as scp, however, gridFTP supports a wide variety of configuration options to tune its performance over networks with different characteristics.

gridFTP is often used as part of some larger system. One such system typical in OSG is the Storage Resource Manager (SRM). [8] SRM queues up transfer requests to a site. It then dispatches them to a network of gridFTP servers, effectively implementing site-level load balancing and network throttling. Another such system is Globus Online (GO) [9], which automatically resumes failed transfers, optimizes the gridFTP parameters to minimize transfer time, implements community-level load balancing of gridFTP servers, and provides an easy mechanism ("Globus Connect" [10]) to install a transfer client on a user's PC.

Another method to transfer data is with http, possibly through a SQUID proxy. [11] The SQUID proxy can automatically cache input files at a site, reducing the overall network traffic, although it does not have a mechanism to throttle the transfers.

2.3. Site Storage and Data Management

The following storage options are typically available to a job:

- The local disk on the worker node where the job is running: Condor usually transfers data to this storage location. In OSG, this space should be at least 10GB and is typically more.
- A shared file system for moderate amounts of data: These are accessible to jobs through environment

variables such as $OSG_APP and $OSG_DATA [12] via a POSIX interface. From outside the site, these areas are accessible through gridFTP or SRM interfaces. This space is intended mainly for pre-staging data before the jobs run.

- A Storage Element [13]: this is often deployed as a disk-based storage with an SRM interface and, sometimes, a POSIX interface for internal access. At large sites, the deployment is typically more complex, supporting petabyte-sized areas with tape-backed storage and community-oriented usage policies. A typical use case for this space is storing the large amounts of output data from the jobs. Note that data stored by opportunistic VOs here may eventually be deleted to make room for more.

It is possible to run data transfer jobs or to directly use the data transfer methods discussed above to move data to these locations; however, the management of such transfers can soon become very complicated when a lot of sites are involved. To automate these processes, some communities have adopted data management software such as the *OSG Match Maker (OSGMM)* [14]. OSGMM would periodically use Condor to run data synchronization tasks at all sites of interest to the community. Other large communities with data-intensive needs instead rely on community-specific data management systems that interact with OSG storage through the standard interfaces.

3. Adapting an Application for OSG

When developing a workflow for an application to run on OSG, there are some common considerations and limitations to note. [1]

3.1. Application Portability

The operating system on the nodes varies from site to site but is typically Scientific Linux 5 (SL5) for 64 bit architectures, with SL6 emerging as an alternative. Typically, applications can rely only on the standard libraries that come with the operating system. Any non-standard library must be sent with the job, or the jobs must be restricted to run on only those worker nodes with the libraries.

Applications may run from different directories at different sites so they should use relative paths or standard OSG environment variables (such as $OSG_APP and $OSG_DATA – section 2.3) to locate any files that they need. Also, the application should be ready to handle file system paths that may be much longer than on non-grid computers.

Finally, applications that rely on pre-staging executable to sites should assume that the distribution directories are read-only. Applications that use the distribution areas as temporary scratch space cannot properly work in a cluster environment where multiple applications instances run concurrently.

3.2. Job Interruption

Often, by policy a batch system interrupts a job after one or two days of continuous running. This is typically referred to as *eviction.* To maximize the probability of completion, we recommend running jobs for less than 12 hours. Some sites offer batch system queues that have longer time deadlines and make them available for opportunistic use.

Sites make available opportunistic cycles under the premise that the communities owning the resources are sometimes underutilizing them. These idle cycles are made available to OSG for the benefit of the consortium. The priority of a job running opportunistically is always lower than that of the jobs of communities who own the resources. When higher priority jobs are submitted, irrespectively of how long the opportunistic job has been running, it is generally interrupted immediately or within a day. This effect is referred to as *pre-emption*. [30]

By default, glideinWMS automatically resubmits jobs that are interrupted, typically to a different site. For this to work, jobs running on opportunistic cycles need to run without side effects – like modifying a database – or to be able to reverse them. Running opportunistically gives access to additional resources, but requires special arrangements to support long execution times. For example, by using queues that allow running long jobs.

3.3. Other Resource Constraints

OSG users need to be cognizant of a few more constraints of OSG resources. [15]

OSG implements a DHTC model, whereby communications between processes tend to be high-latency. Heavily parallel computations are restricted to run on single multi-core worker nodes.

The RAM available to each job is about 2 GB on the typical worker node. This limitation is typically not strictly enforced, but jobs that go over the quota may crash the node or end up being evicted.

Local scratch space is limited to a minimum of 10 GB per slot. Jobs are encouraged to clean up after execution.

3.4. Concurrent Application Instances

Scaling up an application to run many simultaneous instances on the grid requires more attention to designing and implementing a workflow. The following workflow metrics are especially important:

- Aggregate wall time: the User Support group coordinates activities among opportunistic users. The total execution time of a computational campaign is an important parameter to identify a potential shortage of opportunistic cycles.
- Aggregate data transfer: the amount of data to transfer for input, output, and executables tends to dictate the type of storage strategy for the given workflow.
- Number of steps in workflow: for some applications, increasing the number of workflow steps can reduce the calendar time that each job needs, thus fitting the workflow within the typical time limitation at sites.
- For complex workflows, good bookkeeping becomes crucial. This includes keeping track of input files processed, jobs failed that need resubmission after fixing the application, and output data location. The bookkeeping can be the most difficult part of solving a particular problem.

4. Individual Projects

This section describes the workflow developed in the past two years for some new communities using the OSG. The projects are roughly in order from the simplest to the most complicated workflow.

The following figure and table is intended to give a sense of scale of the possible workflows. Similar values of these metrics should ideally lead to similar workflows but, as discussed in each section, the workflows may be different because we did not know all their limitations at the time. For comparison, we also show data for US CMS and for D0's runs on OSG. Their data is for one calendar day of operations, which puts them on the same scale as the other OSG projects. Note, though, that the OSG projects accumulated their statistics over the course of months.

Project	Workflow Steps	Job Count	Wall time (h)	Data (TB)	Hours per job
Pheno at SLAC	1	9,000	100,000	1.9	11
EIC	1	158,000	599,000	3	4
LSST Simulation	380	380,000	909,000	5	7
NEES/OpenSees	1	17,000	509,000	12	29
DES (1 day)	1	300	5,000	5.4	16
US CMS (1 day)	10	102,000	519,000	50	5
D0 (non-local) (1 day)	1	18,000	130,000	1	7

Table 1. Table of computational metrics from Section 3.4 for some OSG communities. The time and data transfer numbers are *estimates* from smaller scale tests, the users logs, checking the OSG Gratia Server, and discussions with representatives from experiments.

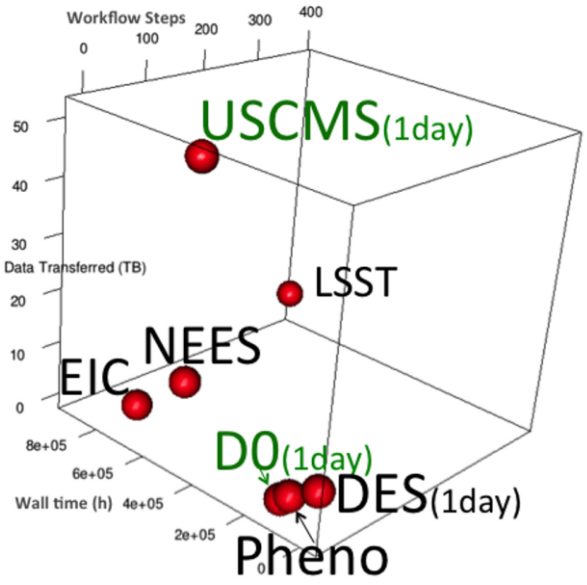

Figure 1: A graphical representation of the data in table 1.

4.1. EIC

The Electron Ion Collider (EIC) is a proposed facility at the Brookhaven National Laboratory for studying the structure of nuclei. To prepare the accelerator design, physicists are using an event generator for Electron/Ion collisions, which requires a pre-calculated table of collision amplitudes. The target of this production campaign on the Grid is calculating these amplitudes, a computationally challenging task. [16]

4.1.1. Steps in workflow. The EIC workflow is a fairly straightforward use of Condor and glideinWMS. The jobs were short, about 4 hours each, so evictions and pre-emptions did not occur frequently. The main challenge was making available a large file, about 1 GB, to each job.

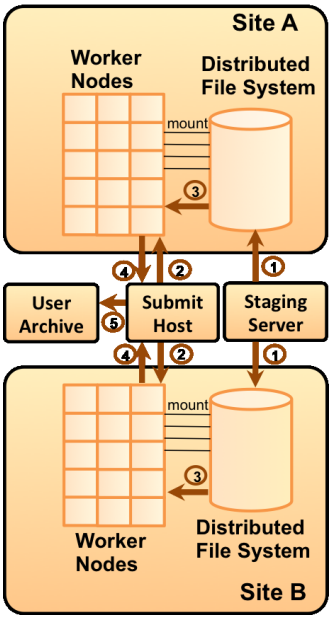

Figure 2: Illustration of a simple workflow with pre-staged data. The numbers here correspond to the steps explained in the text below

1. (Only once) Pre-stage a 1GB read-only file to each site's $OSG_DATA area. This way, that file does not need to be repeatedly transferred over the wide area network.
2. Submit the jobs to the sites with the pre-staged files. Use Condor to transfer the application with the job.
3. Jobs run and read the pre-staged files.
4. Condor transfers the output data back to the submit host.
5. User downloads the output data to their local storage.

4.1.2. Practical Considerations. The file in Step 1 of the workflow is needed by every job. We use an OSGMM server to pre-stage this file. We also arrange for the glideinWMS pilot job [17] to indicate which sites have the data, so that jobs can run only at those sites.

Pre-staging files to $OSG_DATA saves much redundant data movement from the submit host, but there will still be a lot of network traffic at each site. Another disadvantage is that users need help from staff to set up OSGMM and glideinWMS. The *CVMFS* system may solve both of these problems in the future. Once a file is downloaded at a worker node, that file may be kept in a local cache for job access. CVMFS offers a POSIX interface to the file through FUSE. [18]

These jobs require a lot of hours in aggregate, but it was possible to tune the application to run many jobs of short duration. This fits well with the typical constraints on job duration in the OSG environment.

4.2. NEES

Members of the Network for Earthquake Engineering Simulation (NEES)[19] can use the OpenSees application to simulate the effects of earthquakes on building structures. OpenSees was developed by the Pacific Earthquake Engineering Research Center. [20]

This project involved studies of the 13-story National Earthquake Hazards Reduction Program (NEHRP) building model. In particular, the computational campaign did a probabilistic seismic demand hazard analysis of the building and studied how the Finite Element model parameters affect the analysis. [21]

4.2.1. Steps in workflow. Each job requires a small amount of input data (~60MB) and produces about 1.5GB of output data on average. To handle this, we chose a simple Grid workflow with Condor handling jobs and I/O followed by the use of Globus Online (GO) to transfer the data produced to the user data archive. In detail:

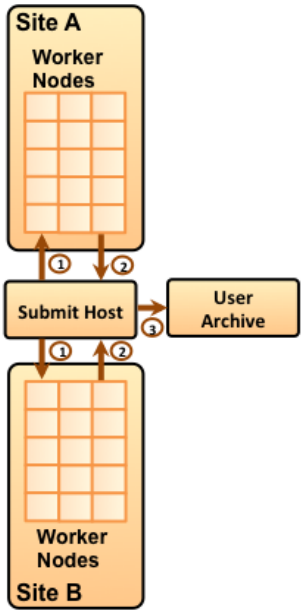

Figure 3: Diagram of NEES workflow.

1. Use Condor/glideinWMS to submit the OpenSees simulation application to sites. Condor transfers the input data and then starts the jobs running.
2. Use Condor to return the data to the submit host.
3. Use GO to transfer the data back to the user's archive.

This workflow is actually simpler than the one for EIC, but had some difficulties as discussed below.

4.2.2. Practical Considerations. The aggregate amount of output data turned out to be larger than originally expected, which caused several problems:

- The submit host became overwhelmed by the large number of simultaneous transfers of data back from the worker nodes. The solution for this was to adjust condor to restrict the number of simultaneous transfers.
- Data transfer was hindered by a faulty network cable. Eventually, the site administrators found and fixed this. We used basic network utilities (such as iperf) to expose the problem.
- There was not enough disk space on the submit host. The administrators agreed to install more.
- The user initially had external hard disks as a data archive. Their USB interfaces were not fast enough to keep up with the rate of data transferred over the network. The user ended up installing internal drives.

Fixing all of these issues made the above workflow successful enough to carry out the research. The workflow for the Phenomenology project discussed below, however, would have been more appropriate workflow for this case.

Another problem was that the OpenSees jobs could sometimes execute for longer than a day – forty hours – and experienced many preemptions and evictions. We experimented with ways of directing jobs to less busy sites, with mixed results.

4.3. SLAC Phenomenology Research

The phenomenology group at SLAC ran an application called Sherpa [22] that does multi-particle quantum chromodynamics calculations using Monte Carlo methods. One result of this work was performing test runs for simulations that will help in searches for new physics. [23]

4.3.1. Steps in workflow. We implemented a workflow with the following steps:

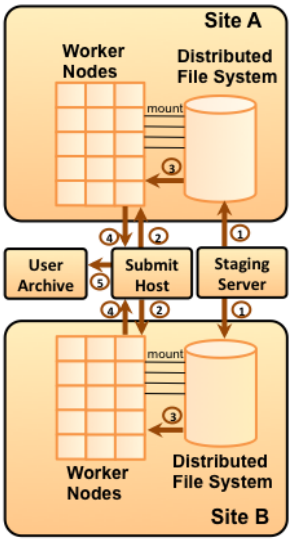

Figure 4: Diagram of SLAC Phenomenology workflow.

1. (Only once) Stage software to each site's $OSG_APP area.
2. Use Condor/glideinWMS to submit the jobs to the sites and transfer input data to the scratch space on the worker nodes. When each job is done Condor transfers output data to the local Storage Element at the site where it ran. It does this using either an SRM [24] or POSIX (the unix *cp* command) interface.
3. When the runs are finished, users move the output data directly from the Storage Elements at each site to their local archive.

4.3.2. Practical Considerations. The main feature of this workflow is that there can be a fairly large amount of output data produced. After the experience with OpenSees, we decided to use the distributed workflow described above, where the data from the jobs that run at each site stays at that site until the user later downloads it at a reasonable rate. This spreads out the load compared to returning all the data to just a single server. An additional feature of this workflow is that SRM will only process as many requests as it can handle, and will queue up the rest, protecting the hosting machine from system crashes. [25]

The current workflow is implemented with custom scripts. Instead it may be easier to use iRODS [26] to manage this data storage and movement.

4.4. DES

To better understand the properties of dark energy, the Dark Energy Survey (DES) is planning to collect and process images of the night sky. [27][28]

The DES workflow has run only on a small scale on the Grid, but larger runs are being prepared. Currently a workflow similar to the one used for Phenomenology is under consideration: worker nodes produce data which they then move to a storage element at the site. Then a separate step moves the data to the experiment's long-term storage.

4.4.1. Practical Considerations. Each job needs to transfer a lot of data for input and output. If many jobs start or finish at once then this could overwhelm the network and storage bandwidth. It should help to stagger the job starts to reduce peaks in the transfer rates.

The processing for one exposure may take longer than the application requirements allow. This may require simultaneously processing different parts of the exposure, which would complicate the workflow and would require transferring even more data.

4.5. LSST Simulation

The Large Synoptic Survey Telescope (LSST) will image a large area of the sky with each exposure. This will help it in accomplishing its science missions [29,p.2]:

1. Probing dark energy and dark matter
2. Taking an inventory of the solar system
3. Exploring the transient optical sky
4. Mapping the Milky Way

One of the main features of the telescope is a 3.2 Gigapixel camera, which is anticipated to produce about 15TB of uncompressed image data a night. Another is its large mirror, which allows the telescope to quickly detect faint objects in a large area of the sky. [29]

The User Support team helped the LSST collaboration port their image simulation application to the OSG. Simulated images are used to refine and validate the data analysis software used to accomplished LSST scientific missions. [29, Section 2.7] For one exposure the software simulates the path of 10^{11} photons from their sources, through the atmosphere, the telescope optics, and to the CCDs.[31]

4.5.1. Steps in workflow. At a high-level, the LSST simulation of one image consists of the aggregation of 189 images, one for each camera chip, each rendered independently from one another. More in detail, for every simulated condition of the telescope (position, time of the observation, etc.), the software produces two images simulating two exposures of the camera (189 x 2 chip images).

The input to the simulation consists of information common to all jobs, namely a star catalog and the focal plane configuration, and information specific to the simulated image, such as the direction of the telescope, the speed of the wind, position of the moon, etc. The image-specific information consists of 500 MB of data and the common information of 15 GB, which we pre-installed at each site.[2]

To run this simulation in the OSG environment, we mapped its computational steps to the following workflow:

1. (Only once) Pre-stage star catalog and focal plane configuration files to OSG sites (15 GB).
2. Submit 1 job to trim the pre-staged catalog file into 189 files, one per chip in the camera.
3. Submit 2 x 189 jobs: simulate 1 image pair (same image with 2 exposures). Transfer "instance catalog" (telescope position, wind speed, etc.) with each job.
4. Gather output, perform bookkeeping, etc. The expected output was a compressed image of 25 MB per chip.

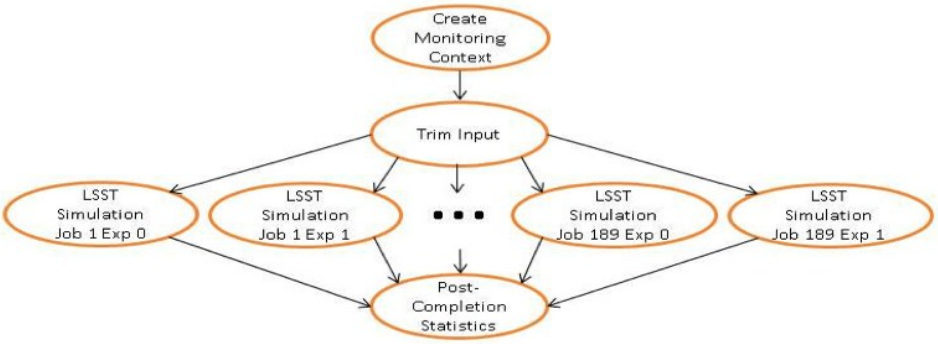

Figure 5: Example of the LSST workflow.

The production campaign consisted in the simulation of one night of data collection i.e. 500 image pairs (2 camera exposures per image). With our workflow, this resulted ideally in 200,000 simulation jobs (one chip per job) plus 500 trim jobs. Overestimating an average of 4 hours per job, this leads to a campaign requiring 800,000 CPU hours of computation. With 2000 cores of available opportunistic resources (50,000 CPU hours/day), the campaign was estimated to take 17 days, every day producing 31 image pairs, moving 50 GB of input and 300 GB of output for a total of 5 TB and 400,000 files.

4.5.2. Practical Considerations. The LSST binaries were not ready for running on the Grid. We had to work with the LSST simulation team to address the following issues

- The application assumed a writable software distribution, not supported on the Grid.
- The application assumed path-lengths too short for the Grid environment.
- The orchestration script was not properly propagating error codes from the programs it managed.

The typical failures at sites resulted from jobs requiring more memory than the batch system allotted and from the storage being unavailable due to maintenance at some of the most productive sites. The disk quota at the submission machine was initially limited.

Despite these challenges, the actual timeline and campaign characteristics were not significantly different from the theoretical ones discussed above.

The LSST workflow distinguished itself mainly because of the complexity of the workflow stemming from the need to preprocess (trim) the job input. The pre-staging of the large catalog file to all sites was also a challenge that the new version of the workflow run at Purdue has overcome.

To validate our results, we compared the images simulated on the Grid with references produced by the standard production mechanisms of the LSST simulation team. 99% of the images were identical, and the differences on the remaining 1% (14 chips) could be explained by issues in the comparison process. All in

[2] The information necessary for the simulation of each job is now extracted before running the workflow and is shipped with every job, rather than being pre-installed at each site.

all, the production campaign was a success and popularized the strategy of using OSG resources within the LSST community.

5. Conclusions

The Open Science Grid User Support team assists new communities in porting their applications to run on OSG. By carefully choosing workflows, it is possible to successfully handle computations from a wide variety of fields on the OSG. These can require fairly large amounts of CPU time and data transfer. By providing access to its opportunistic resources, OSG fosters the production of scientific knowledge at minimal cost to the communities in the consortium.

With more experience and with the maturing of the middleware we hope that the range of problems that are straightforward to run on OSG continues to increase.

6. Acknowledgments

Fermilab is operated by the Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

The Open Science Grid (http://www.opensciencegrid.org) is funded by the National Science Foundation and the Office of Science, Department of Energy (DOE).

We're grateful to help from Oliver Gutsche, Stefan Hoeche, Burt Holzman, Adam Lyon, Derek Weitzel, the system administrators at RENCI and UCSD, the DES Data Management group, and all our users.

7. References

[1] "What is the Open Science Grid?", https://twiki.grid.iu.edu/bin/view/Documentation/WhatIsOSG, rev. 37, accessed 5/31/12.

[2] "An Introduction for Grid Users", https://twiki.grid.iu.edu/bin/view/Documentation/UsingTheGrid, rev. 65, accessed 5/31/12

[3] W. Chen, E. Deelman, "Partitioning and Scheduling Workflows across Multiple Sites with Storage Constraints", 9th International Conference on Parallel Processing and Applied Mathematics (PPAM 2011), Poland, Sep 2011. http://www.isi.edu/~wchen/papers/spc-final.pdf

[4] "Grid resource allocation manager", http://en.wikipedia.org/wiki/Grid_resource_allocation_manager, accessed 5/31/12

[5] "Condor Web Page", http://research.cs.wisc.edu/condor/, accessed 5/31/12

[6] I. Sfiligoi, "glideinWMS - A generic pilot-based Workload Management System" 2008 J. Phys.: Conf. Ser. 119 062044, http://iopscience.iop.org/1742-6596/119/6/062044

[7] “gridFTP” , http://www.globus.org/toolkit/docs/latest-stable/gridftp/, accessed 5/31/12

[8] “Storage Resource Manager”, http://en.wikipedia.org/wiki/Storage_Resource_Manager", 12/17/11, accessed 5/31/12

[9] “Globus Online”, https://www.globusonline.org/, accessed 5/31/12

[10] “Globus Connect”, https://www.globusonline.org/globus_connect/, accessed 5/31/12

[11] "Using HTTP on the OSG", https://twiki.grid.iu.edu/bin/view/Documentation/OsgHttpBasics, rev. 20, accessed 5/31/12

[12] "Local Storage Configuration", https://twiki.grid.iu.edu/bin/view/ReleaseDocumentation/LocalStorageConfiguration, rev. 54, accessed 5/31/12

[13] "Storage Infrastructure Software", https://twiki.grid.iu.edu/bin/view/Documentation/StorageInfrastructureSoftware, rev.43, accessed 5/31/12

[14] "OSG MM - The Open Science Grid Match Maker", http://osgmm.sourceforge.net/, accessed 5/31/12

[15] "Helpful Hints for Running At-Scale on the OSG", https://twiki.grid.iu.edu/bin/view/Documentation/GoldenRulesForOSGUsage , rev. 14, accessed 5/31/12

[16] Tobias Toll, "Electron Ion Collider Simulations on OSG", https://twiki.grid.iu.edu/bin/view/Management/NovDec2011, accessed 5/31/12

[17] "GlideinWMS – Custom Scripts", http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/doc.prd/factory/custom_scripts.html, accessed 5/31/12
[18] P. Buncic, "CernVM – a virtual software appliance for LHC applications", Journal of Physics: Conference Series 219 (2010), CHEP 2009, http://iopscience.iop.org/1742-6596/219/4/042003/pdf/1742-6596_219_4_042003.pdf
[19] "NEES Overview", http://nees.org/aboutnees/overview, accessed 5/31/12
[20] "The Open System for Earthquake Engineering Simulation", http://opensees.berkeley.edu/OpenSees/home/about.php, accessed 5/31/12
[21] A. Barbosa, J. Conte, J. Restrepo, "Running OpenSees Production for NEES on OSG", https://twiki.grid.iu.edu/bin/view/Management/Oct2011Newsletter, accessed 5/31/12
[22] "Sherpa", http://www.sherpa-mc.de/, accessed 5/31/12
[23] S. Höche, F. Krauss, M. Schönherr, F. Siegert, "W+n-jet predictions with MC@NLO in SHERPA", SLAC-PUB-14859,IPPP-12-03,DCPT-12-06,LPN12-026,MCNET-12-01,FR-PHENO-2012-001, http://inspirehep.net/record/1086175
[24] "Storage Resources", https://twiki.grid.iu.edu/bin/view/Documentation/Release3.NavAdminStorage, rev. 35, accessed 5/31/12.
[25] I. Bird et al, "SRM Joint Functional Design Summary of Recommendations", https://sdm.lbl.gov/srm-wg/doc/SRM.Joint.Functional.Design.Jan2002.pdf, accessed 5/31/12
[26] "IRODS:Data Grids, Digital Libraries, Persistent Archives, and Real-time Data Systems", https://www.irods.org/index.php/IRODS:Data_Grids,_Digital_Libraries,_Persistent_Archives,_and_Real-time_Data_Systems, accessed 5/31/12
[27] The Dark Energy Survey website, http://www.darkenergysurvey.org/, accessed 5/31/12
[28] "The Data Management System", http://www.darkenergysurvey.org/DECam/data-manage.shtml, accessed 5/31/12
[29] Ž. Ivezić et al., "LSST: From Science Drivers to Reference Design and Anticipated Data Products", http://arxiv.org/abs/0805.2366, version 2.0.9 of June 4, 2011.
[30] "Brief Analysis on Preemption Handling in OSG", https://twiki.grid.iu.edu/bin/view/VirtualOrganizations/PreemptionHandling_OSG, rev. 3, accessed 5/31/12
[31] "LSST Image Simulation", http://lsst.astro.washington.edu/intro/overview/ , accessed 6/1/12